42P14183 *PATENT*

UNITED STATES DESIGN PATENT APPLICATION

FOR

USE OF MULTIPLE VIRTUAL MACHINE MONITORS TO HANDLE PRIVILEGED EVENTS

Inventors:

GILBERT NEIGER

STEPHEN M. BENNETT

ALAIN KAGI

STALINSELVARAJ JEYASINGH

ANDREW V. ANDERSON

RICHARD UHLIG

ERIK COTA-ROBLES

SCOTT RODGERS

LAWRENCE SMITH

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard, Seventh Floor
Los Angeles, CA 90025-1026

(408) 720-8300

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EV336588666US
Date of Deposit September 15, 2003
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Michelle Begay
(Typed or printed name of person mailing paper or fee)

Michelle Begay
(Signature of person mailing paper or fee)

USE OF MULTIPLE VIRTUAL MACHINE MONITORS TO HANDLE PRIVILEGED EVENTS

Field of the Invention

[0001] The present invention relates generally to virtual machines, and more specifically to handling privileged events using multiple virtual machine monitors.

Background of the Invention

[0002] A conventional virtual-machine monitor (VMM) typically runs on a computer and presents to other software the abstraction of one or more virtual machines. Each virtual machine may function as a self-contained platform, running its own "guest operating system" (i.e., an operating system (OS) hosted by the VMM) and other software, collectively referred to as guest software. The guest software expects to operate as if it were running on a dedicated computer rather than a virtual machine. That is, the guest software expects to control various events and have access to hardware resources. The hardware resources may include processor-resident resources (e.g., control registers), resources that reside in memory (e.g., descriptor tables) and resources that reside on the underlying hardware platform (e.g., input-output devices). The events may include internal interrupts, external interrupts, exceptions, platform events (e.g., initialization (INIT) or system management interrupts (SMIs)), execution of certain instructions, etc.

[0003] In a virtual-machine environment, the VMM should be able to have ultimate control over these events and hardware resources to provide

proper operation of guest software running on the virtual machines and for protection from and between guest software running on the virtual machines. To achieve this, the VMM typically receives control when guest software accesses a protected resource or when other events (such as interrupts or exceptions) occur. For example, when an operation in a virtual machine supported by the VMM causes a system device to generate an interrupt, the currently running virtual machine is interrupted and control of the processor is passed to the VMM. The VMM then receives the interrupt, and handles the interrupt itself or delivers the interrupt to the appropriate virtual machine.

Brief Description of the Drawings

[0004] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0005] **Figure 1** illustrates one embodiment of a virtual-machine environment, in which the present invention may operate;

[0006] **Figures 2** illustrates a configuration of multiple VMMs in a virtual-machine environment;

[0007] **Figure 3** is a flow diagram of one embodiment of a process for handling privileged events in a virtual-machine environment having multiple VMMs;

[0008] **Figures 4, 6 and 7** illustrate exemplary embodiments of processes of identifying a VMM for handling a privileged event; and

[0009] **Figure 5** illustrates an exemplary usage of process 400 in a virtual machine environment having two VMMs.

Description of Embodiments

[0010] A method and apparatus for handling privileged events using multiple virtual machine monitors are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention can be practiced without these specific details.

[0011] Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer system's registers or memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0012] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically

stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or the like, may refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer-system memories or registers or other such information storage, transmission or display devices.

[0013] In the following detailed description of the embodiments, reference is made to the accompanying drawings that show, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention. Moreover, it is to be understood that the various embodiments of the invention, although different, are not necessarily mutually exclusive. For example, a particular feature, structure, or characteristic described in one embodiment may be included within other embodiments. The following detailed description is, therefore, not to be taken in a limiting sense, and the

scope of the present invention is defined only by the appended claims, along with the full scope of equivalents to which such claims are entitled.

[0014] Although the below examples may describe embodiments of the present invention in the context of execution units and logic circuits, other embodiments of the present invention can be accomplished by way of software. For example, in some embodiments, the present invention may be provided as a computer program product or software which may include a machine or computer-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. In other embodiments, steps of the present invention might be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

[0015] Thus, a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Disc, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, a transmission over the Internet, electrical, optical, acoustical

or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.) or the like.

[0016] Further, a design may go through various stages, from creation to simulation to fabrication. Data representing a design may represent the design in a number of manners. First, as is useful in simulations, the hardware may be represented using a hardware description language or another functional description language. Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. Furthermore, most designs, at some stage, reach a level of data representing the physical placement of various devices in the hardware model. In the case where conventional semiconductor fabrication techniques are used, data representing a hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. In any representation of the design, the data may be stored in any form of a machine-readable medium. An optical or electrical wave modulated or otherwise generated to transmit such information, a memory, or a magnetic or optical storage such as a disc may be the machine readable medium. Any of these mediums may "carry" or "indicate" the design or software information. When an electrical carrier wave indicating or carrying the code or design is transmitted, to the extent that copying, buffering, or re-transmission of the electrical signal is performed, a new copy is made. Thus, a communication provider or a

network provider may make copies of an article (a carrier wave) embodying techniques of the present invention.

[0017] **Figure 1** illustrates one embodiment of a virtual-machine environment 100, in which the present invention may operate. In this embodiment, bare platform hardware 110 comprises a computing platform, which may be capable, for example, of executing a standard operating system (OS) or virtual-machine monitors (VMMs), such as VMMs 125.

[0018] The platform hardware 110 can be of a personal computer (PC), mainframe, handheld device, portable computer, set-top box, or any other computing system. The platform hardware 110 includes at least one processor 112, memory 120 and possibly other platform hardware (e.g. input-output devices), not shown.

[0019] Processor 112 can be any type of processor capable of executing software, such as a microprocessor, digital signal processor, microcontroller, or the like. The processor 112 may include microcode, programmable logic or hardcoded logic for performing the execution of method embodiments of the present invention.

[0020] Memory 120 can be a hard disk, a floppy disk, random access memory (RAM), read only memory (ROM), flash memory, any combination of the above devices, or any other type of machine medium readable by processor 112. Memory 120 may store instructions or data for performing the execution of method embodiments of the present invention.

[0021] Each VMM 125, though typically implemented in software, may emulate and export a bare machine interface to higher level software. Such higher level software may comprise a standard or real-time OS, may be a highly stripped down operating environment with limited operating system functionality, may not include traditional OS facilities, etc. The VMMs 125 may be implemented, for example, in hardware, software, firmware, or by a combination of various techniques.

[0022] When running, each VMM 125 presents to "guest" software (i.e., software other than that of the VMMs 125) the abstraction of one or more virtual machines (VMs). The VMMs 125 may provide the same or different abstractions to the various guests. The guest software running on each VM may include a guest OS (e.g., a guest OS 134, 144 or 154) and various guest software applications (e.g., applications 136, 146 and 156). Collectively, guest OS and software applications are referred to herein as guest software 103, 105 and 115.

[0023] Guest software 103, 105 and 115 expects to access physical resources (e.g., processor registers, memory and I/O devices) within the VMs 132, 142 and 152 on which the guest software is running. The VMMs 125 facilitate access to resources desired by guest software while retaining ultimate control over resources within the platform hardware 110. In addition, the guest software 103, 105 and 115 expect to handle various events such as exceptions, interrupts and platform events (e.g., initialization (INIT) and system management interrupts (SMIs)). Some of these events are

"privileged" because they cannot be handled by the guest software to ensure proper operation of VMs 132, 142 and 152 and guest software 103, 105 and 115 and for protection from and between guest software 103, 105 and 115. The privileged events are handled by the VMMs 125.

[0024] In particular, a specific VMM is identified to handle each privileged event. In one embodiment, a specific VMM is identified using routing logic 130. In some embodiments, the routing logic 130 is implemented as microcode, programmable logic or hardcoded logic. In other embodiments, the routing logic 130 is implemented as software residing in memory 120. In yet other embodiment, the routing logic 130 is implemented as a combination of hardware and software.

[0025] Once a specific VMM is identified, it will facilitate functionality desired by guest software while retaining ultimate control over this privileged event. The act of facilitating the functionality for the guest software may include a wide variety of activities on the part of the VMMs 125. These activities of the VMMs 125 should not limit the scope of the present invention.

[0026] The system 100 may include two or more VMMs 125 executing on the platform hardware 110. In one embodiment, the VMMs 125 run in parallel, and each VMM 125 can receive control from a VM. **Figure 2** illustrates one embodiment of such configuration of the VMMs 125 within the system 100.

[0027] Referring to **Figure 2**, an exemplary parallel configuration of VMMs 210 through 214 is illustrated. According to this configuration, control transfers from a VM to a specific VMM when a privileged event occurs during the operation of a VM. The specific VMM is identified based on certain criteria as will be discussed in more detail below. Once the specific VMM is identified, the information pertaining to the virtualization event is delivered to this VMM, and control is transitioned to it. For example, privileged event 1 occurring during the operation of VM 202 may cause transition from VM 202 to VMM 210, privileged event 2 occurring during the operation of VM 202 may cause transition from VM 202 to VMM 212, privileged event 3 occurring during the operation of VM 206 may cause transition from VM 206 to VMM 212, and privileged event 4 occurring during the operation of VM 206 may cause transition from VM 206 to VMM 214. A corresponding VMM then handles the virtualization event and may transfer control back to a VM from which the control was received. In one embodiment, the transfer of control from a VMM to a VM is achieved by executing a special instruction. The transfer of control from a VMM to a VM is referred to herein as a VM entry and the transfer of control from a VM to a VMM is referred to herein as a VM exit.

[0028] In one embodiment, when a VM exit occurs, control is passed to a VMM at a specific entry point (e.g., an instruction pointer value) delineated in a virtual machine control structure (VMCS) that resides in memory and is maintained by the processor. In another embodiment, control

is passed to a VMM after vectoring through a redirection structure (e.g., the interrupt-descriptor table in the processor instruction set architecture (ISA) of the Intel® Pentium® 4 (referred to herein as the IA-32 ISA)). Alternatively, any other mechanism known in the art can be used to transfer control from a VM to a VMM.

[0029] A privileged event may occur during the operation of a VMM. Examples of privileged events that may occur during the operation of a VMM may include, for example, system management interrupts (SMIs), INIT, non-maskable interrupts (NMIs), hardware interrupts, interprocessor interrupts (IPIs), etc. In one embodiment, if a privileged event occurs during the operation of a VMM, routing logic is employed to identify a VMM designated to handle this privileged event. If the designated VMM is not the VMM that is currently operating, the information pertaining to the privileged event is delivered to the designated VMM, and control is transitioned to it. For example, privileged event 5 occurring during the operation of VMM 210 may cause transition from VMM 210 to VMM 212. VMM 212 then handles the privileged event and may transfer control back to VMM 210. Hence, in interactions between VMM 210 and VMM 212, VMM 210 plays a role of a VM. Accordingly, in one embodiment, a VM exit and VM entry mechanism similar to the one described above with respect to the transfer of control between a VM and a VMM is used to transfer control between the VMMs.

[0030] **Figure 3** is a flow diagram of one embodiment of a process 300 for handling privileged events in a virtual-machine environment having

multiple VMMs. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (such as run on a general purpose computer system or a dedicated machine), or a combination of both.

[0031] Referring to **Figure 3**, process 300 begins with processing logic detecting the occurrence of a privileged event (processing block 302). A privileged event is an event that is not to be handled by the currently running software. Such privileged events may include exceptions, interrupts, platform events, execution of a "privileged" instruction (an instruction whose execution causes a VM exit), etc. A privileged event may occur during the operation of guest software or during the operation of a VMM.

[0032] At processing block 304, processing logic determines which one of the multiple VMMs is to handle the privileged event. The determination may be based on various factors such as characteristics of the privileged event, current values of designated fields modifiable by the VMMs, operational parameters of the VMMs, operational parameters of guest software, etc. In one example, the determination is based on the type of the privileged event as will be discussed in greater detail below in conjunction with **Figure 4**. In another example, the determination is based on current values of control fields configured by one of the multiple VMMs as will be discussed in more detail below in conjunction with **Figure 6**. In yet another example, the determination is based on the evaluation of load and/or usage characteristics of the VMMs, as will be discussed in greater detail below in

conjunction with **Figure 7**. In still another example, the determination is based on the combination of the above factors.

[0033] Once the processing logic determines which VMM is to handle the privileged event, it delivers information pertaining to the privileged event to the designated VMM and transitions control to this VMM (processing block 306). The VMM may then handle the privileged event itself or route it to guest software for handling.

[0034] **Figures 4, 6 and 7** illustrate exemplary embodiments of processes of identifying a VMM for handling a privileged event. The processes may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (such as run on a general purpose computer system or a dedicated machine), or a combination of both.

[0035] Referring to **Figure 4**, process 400 begins with processing logic detecting the occurrence of a privileged event during the operation of guest software or a VMM (processing block 402). At processing block 404, processing logic identifies the type of the privileged event. Next, processing logic determines which VMM is designated to handle privileged events of this type (processing block 406). In one embodiment, each event type is statically mapped to a particular VMM (e.g., using hardcoded logic). In another embodiment, the determination is dynamic, as described below with regard to **Figure 6**. In yet another embodiment, a combination of statically determined VMMs and dynamically determined VMMs is used. That is,

some event types may be statically mapped to corresponding VMMs while other event types may require some additional processor operation for the determination.

[0036] If the currently-operating entity is not the designated VMM (decision box 408), processing logic transitions control to the designated VMM (processing block 410). In one embodiment, the transition to the designated VMM may be accomplished by generating a VM exit. Alternatively, any other mechanism known in the art may be used to transition control to the designated VMM.

[0037] If the currently-operating entity is the designated VMM, the event is delivered to the currently running VMM (processing block 412). In one embodiment, the delivery of the event to the VMM is performed by synthesizing a VM exit from the VMM to itself. In another embodiment, the event is delivered as it would be delivered in a non-virtual machine environment (e.g., by vectoring through an interrupt descriptor table or causing a transition to system management mode). It should be noted that a variety of other mechanisms known in the art may be used to deliver the event to the currently running VMM.

[0038] **Figure 5** illustrates an exemplary usage of process 400 in a virtual machine environment having two VMMs, according to one embodiment of the present invention. One of the two VMMs (e.g., VMM 508) is exclusively designated to handle certain platform events (e.g., system management interrupts (SMIs), IPIs, non-maskable interrupts, etc.). VMM

508 is referred to as the Platform VMM (PVMM). In some embodiments, the PVMM is designated to handle only SMIs. In other embodiments, PVMMs handle additional event types, as well as SMIs. In yet other embodiments, multiple PVMMs are used, each handling different platform events.

[0039] When a privileged event occurs, routing logic 510 determines the type of the event. If the event is a platform event to be handled by the PVMM 508, routing logic 510 routes it to VMM 508. Otherwise, the event is routed to VMM 506. As illustrated in **Figure 5**, routing logic 510 may route events that occur during the operation of guest software or during the operation of a VMM.

[0040] The routing of an event to a VMM may differ depending on what entity was running when the event occurs. If guest software was running, the transition to the VMM selected by the routing logic 510 is performed, in one embodiment, via a VM exit. If the VMM selected by the routing logic was running when the event occurred, the event may be delivered to the VMM through a variety of mechanisms, as described above with regard to **Figure 4**.

[0041] The use of a second VMM designated exclusively to handle certain platform events eliminates product dependency between hardware vendors and OS vendors. That is, it allows platform hardware vendors to develop the code for the second VMM (the PVMM) independently from OS vendors. Similarly, OS vendors can develop the code for the first VMM independently from the hardware vendors. In addition, the use of two VMMs

performing different functionality enhances system security and limits the exposure of the code that needs to be validated for security.

[0042] Referring to **Figure 6**, process 600 begins with processing logic detecting the occurrence of a privileged event during the operation of guest software or a VMM (processing block 602). At processing block 604, processing logic receives information that identifies the privileged event. Next, processing logic accesses a resource (e.g., an in-memory data structure, a processor register, memory in the platform chipset, a register in an input-output device, etc.) that controls the selection of a VMM for handling privileged events (processing block 606) and reads the current value of a resource field associated with the identifier of the privileged event (processing block 608).

[0043] In one embodiment, the identifier of the privileged event is the type of the privileged event, and processing logic uses the type of the privileged event to identify a resource field value associated with this type of privileged event.

[0044] In another embodiment, the identifying information identifies a specific input-output address associated with the privileged event. In this embodiment, processing logic analyzes the input-output access associated with the privileged event, determines what input-output address is being accessed, and then uses this input-output address to find a resource field value associated with an input-output address range to which the determined input-output address belongs. In one embodiment, the values of

the resource fields are set during the initialization and cannot be modified. In another embodiment, the values of the resource fields can be dynamically configured by a VMM (e.g., a main VMM) based on such factors as security requirements, the VMM's knowledge of the platform, etc. This dynamic configuration of resource field values allows for VMM functionality to be partitioned, possibly improving security, system performance, development methodologies, etc.

[0045] Further, processing logic determines which VMM is designated to handle the privileged event based on the current value of the corresponding field (processing block 610) and, if the currently-operating entity is not the designated VMM (decision box 612), transitions control to the designated VMM (processing block 614). In one embodiment, the transition to the designated VMM may be accomplished by generating a VM exit. Alternatively, the transition to the designated VMM may be accomplished through any other mechanism in the art.

[0046] If the designated VMM is the currently-operating entity, processing logic delivers the event to that VMM (processing block 616), as described above in conjunction with **Figure 4** and **Figure 5.**

[0047] Referring to **Figure 7**, process 700 begins with processing logic detecting the occurrence of a privileged event during the operation of guest software or a VMM (processing block 702). At processing block 704, processing logic identifies resource usage/load parameters of each VMM in the system. Next, processing logic evaluates the resource usage and/or load

parameters of the VMMs in the context of load balancing (processing block 706) and determines which VMM is the best candidate for handling the privileged event based on the above load balancing evaluation (processing block 708). Further, if the currently operating entity is not the VMM that was identified as the best candidate (decision box 712), processing logic transitions control to the identified VMM (processing block 714). In one embodiment, the transition to the identified VMM may be accomplished by generating a VM exit. Alternatively, the transition to the identified VMM may be accomplished through any other mechanism in the art.

[0048] If the designated VMM is the currently-operating entity, processing logic delivers the event to that VMM (processing block 716), as described above in conjunction with **Figures 4-6**.

[0049] Thus, a method and apparatus for handling privileged events using multiple VMMs have been described. It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reading and understanding the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.